

EtherCAT Slave Editor

An editor for creating SOES slaves

rt-labs AB

Background

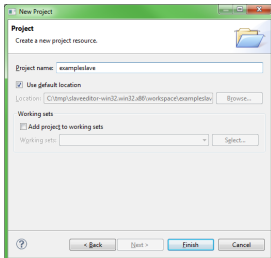
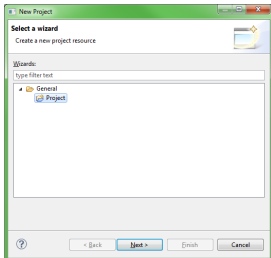
- ▶ When developing an EtherCAT slave it is often tiresome and error prone to update the object dictionary, ESI file and EEPROM and keep them aligned
- ▶ The EtherCAT Slave Editor is a tool for helping out with this process
- ▶ The EtherCAT Slave Editor generates ESI files, EEPROM files, object dictionaries and data structures for slaves using the SOES stack by providing a easy to use interface

Preparation

- ▶ Download the EtherCAT Slave Editor from <http://download.rt-labs.com/ethercat/slaveeditor>
- ▶ Unzip and run
- ▶ The Slave Editor needs the Java Runtime Environment (JRE): <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

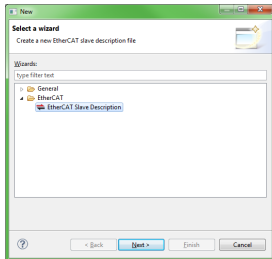
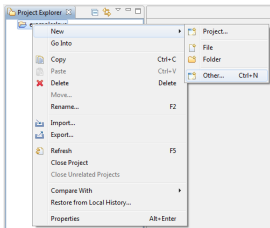
Create a project

- ▶ First we need to create a project
- ▶ Right click in the **Project Explorer**
- ▶ Choose **New / Project**



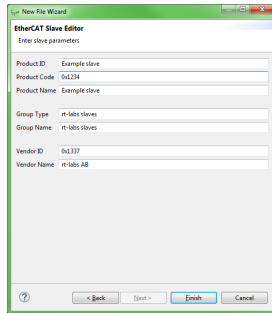
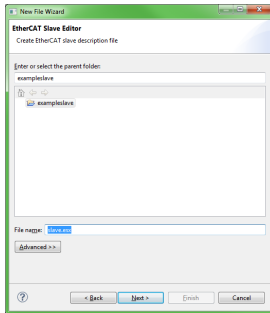
Use the EtherCAT Slave Editor

- ▶ To work with the EtherCAT Slave Editor we need to add a slave description file



Use the EtherCAT Slave Editor

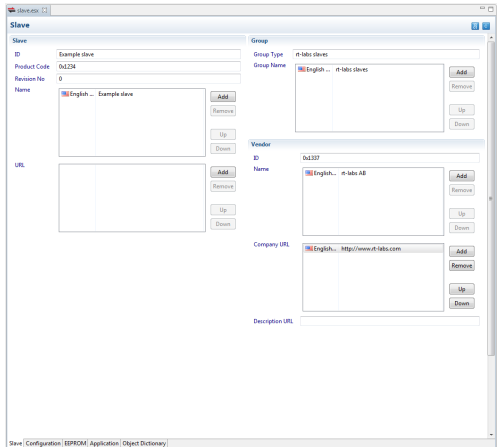
- ▶ Choose a file destination and fill in some basic administrative data
- ▶ Parts of the data will be remembered when creating other slaves



Slave configuration

Administrative data

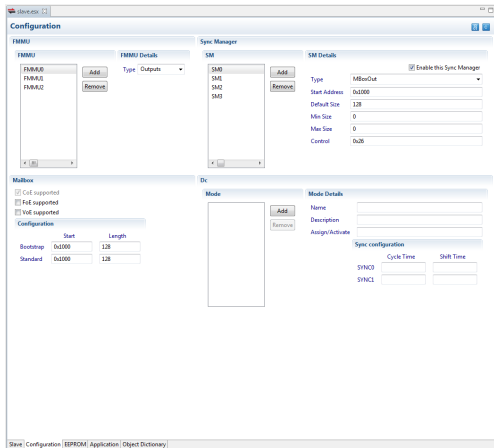
- ▶ This tab contains information previously provided in the wizard



Slave configuration

FMMU/SM/MBOX/DC

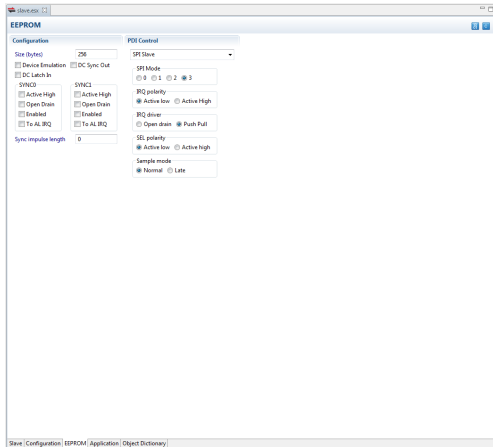
- Parameters for an advance slave with CoE and inputs and outputs are filled in by default



Slave configuration

EEPROM/ESC connection

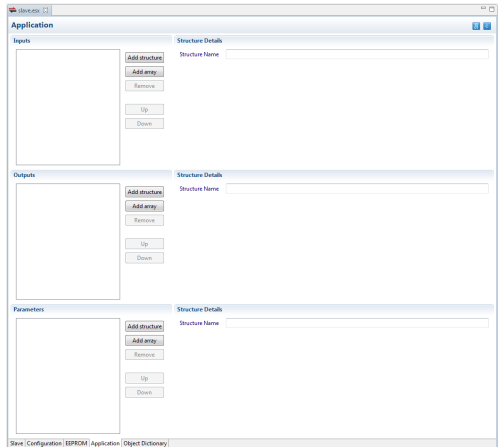
- ▶ This tab contains settings for the EEPROM such as the connection method of the ESC



Slave configuration

Process and service data

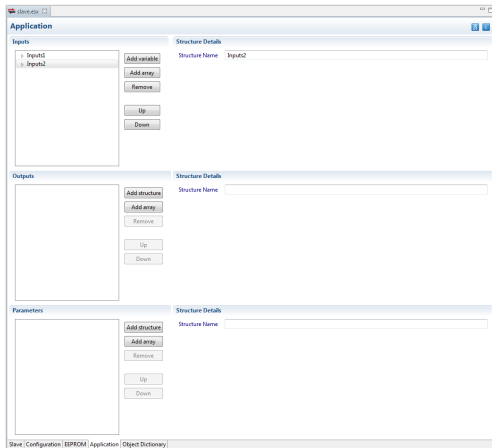
- ▶ In the Application tab we design the interface between the slave and the system
- ▶ In this tab we specify our PDOs and SDOs



Slave configuration

Process and service data

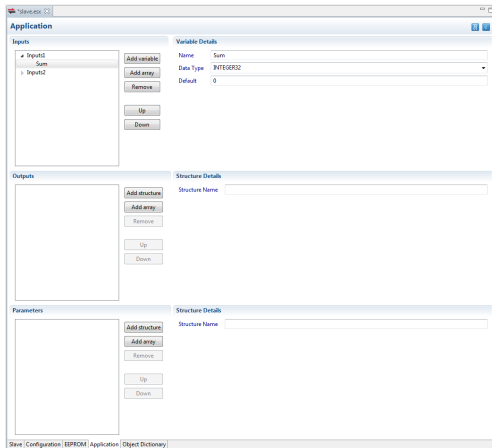
- ▶ We have the possibility to add multiple structures
- ▶ In this case we have created two structures which will later appear at different indexes in the object dictionary
- ▶ Inputs are the data that is sent from the slave



Slave configuration

Process and service data

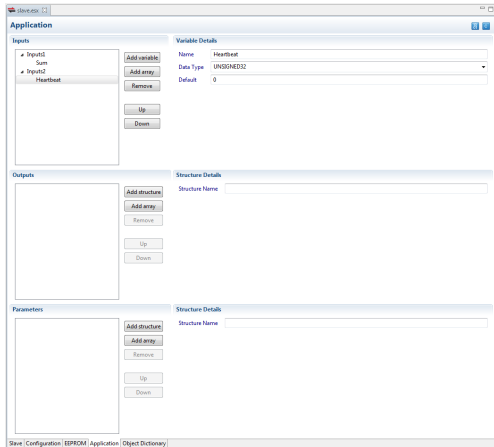
- ▶ Add a variable named **Sum** to the **Inputs1** structure



Slave configuration

Process and service data

- ▶ Add a variable named **Heartbeat** to the **Inputs2** structure



Slave configuration

Process and service data

- ▶ Add an **Output** structure
- ▶ Add two **PDOs** to the **Output** structure

The screenshot shows the 'Slave Configuration' application window. The main title is 'Application'. It is divided into three main sections: Inputs, Outputs, and Parameters. Each section has a list of items and control buttons (Add, Remove, Up, Down). To the right of each list is a 'Variable Details' or 'Structure Details' panel with input fields for Name, Data Type, and Default.

Inputs

- Input1: Sum
- Input2: Heartbeat

Variable Details (for Input2)

Name	Heartbeat
Data Type	UNSIGNED32
Default	0

Outputs

- Output: Value A
- Output: Value B

Variable Details (for Output)

Name	Value B
Data Type	INTEGER32
Default	0

Parameters

- Parameter: (empty)

Structure Details

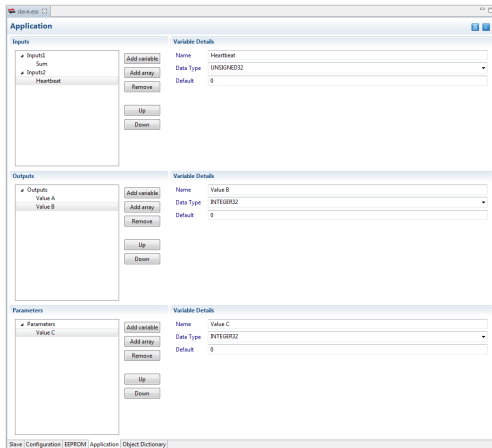
Structure Name	(empty)
----------------	---------

Slave Configuration | EEPROM | Application | Object Dictionary

Slave configuration

Process and service data

- ▶ And add an SDO as well



The screenshot shows the 'Application' configuration window with the following details:

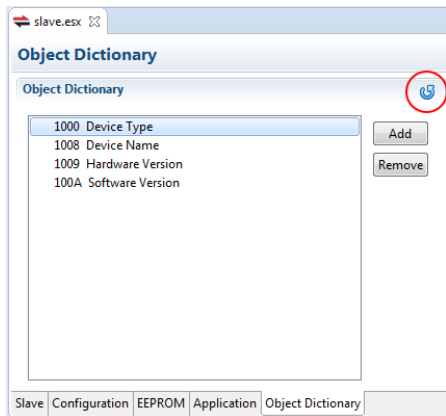
Section	Variable Name	Data Type	Default Value
Inputs	Input1		
	Sum		
	Input2		
	Heartbeat	UNSIGNED32	0
Outputs	Output1		
	Value A		
	Value B	INTEGER32	0
Parameters	Parameter1		
	Value C	INTEGER32	0

The bottom status bar indicates: Slave | Configuration | EEPROM | Application | Object Dictionary

Slave configuration

Object dictionary

- ▶ The Object dictionary tab provides a visualisation of the object dictionary
- ▶ To populate the object dictionary with the data supplied on the other tabs we need to synchronize it



Slave configuration

Populated object dictionary

The screenshot shows a software window titled "slave.exe" with a tab labeled "Object Dictionary". The window is divided into two main sections: "Object Dictionary" on the left and "Object Details" on the right.

Object Dictionary Section:

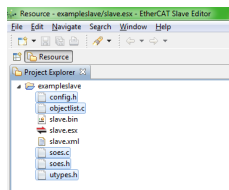
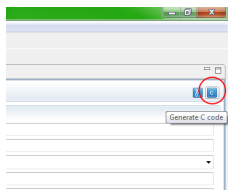
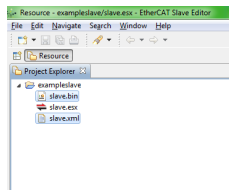
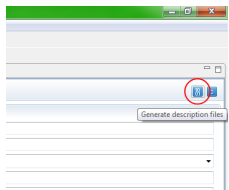
- Buttons: "Add" and "Remove".
- Tree view:
 - 1000 Device Type
 - 1008 Device Name
 - 1009 Hardware Version
 - 100A Software Version
 - 1018 Identity Object
 - 1600 Outputs
 - 1600:00 Number of Elements
 - 1600:01 Value A
 - 1600:02 Value B
 - 1400 Inputs1
 - 1A00:00 Number of Elements
 - 1A00:01 Sum
 - 1401 Inputs2
 - 1A01:00 Number of Elements
 - 1A01:01 Heartbeat
 - 1C00 Sync Manager Communication Type
 - 1C12 Sync Manager 2 PDO Assignment
 - 1C13 Sync Manager 3 PDO Assignment
 - 6000 Inputs1
 - 6000:00 Number of Elements
 - 6000:01 Sum
 - 6001 Inputs2
 - 6001:00 Number of Elements
 - 6001:01 Heartbeat
 - 7000 Outputs
 - 7000:00 Number of Elements
 - 7000:01 Value A
 - 7000:02 Value B
 - 8000 Parameters
 - 8000:00 Number of Elements
 - 8000:01 Value C

Object Details Section:

- Index: 0x01
- Name: Heartbeat
- Data Type: UNSIGNED32
- Default: 0x60010120
- Access: RO

Generate files

EEPROM, ESI, code



Download SOES

- ▶ The generated files could be used with **SOES**
- ▶ **SOES** is available for download from Open EtherCAT Society's GitHub page
- ▶ <https://github.com/OpenEtherCATsociety/SOES>

Implement callback functions

- ▶ All function interfaces are specified in **soes.h**
- ▶ We need to implement the callback functions to handle the PDO and SDO exchanges

```
/**
 * This function reads physical input values and assigns the
 * corresponding members of Rb.Inputs1
 */
void cb_get_Inputs1();

/**
 * This function reads physical input values and assigns the
 * corresponding members of Rb.Inputs2
 */
void cb_get_Inputs2();

/**
 * This function writes physical output values from the
 * corresponding members of Wb.Outputs
 */
void cb_set_Outputs();

/**
 * This function is called after a SDO write of the object
 * Cb.Parameters.
 */
void cb_post_write_Parameters(int subindex);
```

Example implementation of the callback functions

```
void cb_get_Inputs1()
{
    Rb.Inputs1.Sum = Wb.Outputs.Value_A + Wb.Outputs.Value_B + Cb.Parameters.Value_C;
}
```

```
void cb_get_Inputs2()
{
    static int32_t tick = 0;
    Rb.Inputs2.Heartbeat = tick++;
}
```

```
void cb_set_Outputs()
{
}
```

```
void cb_post_write_Parameters(int subindex)
{
    if (subindex == 1)
        printf ("Value_C_written->%d\n", Cb.Parameters.Value_C);
}
```